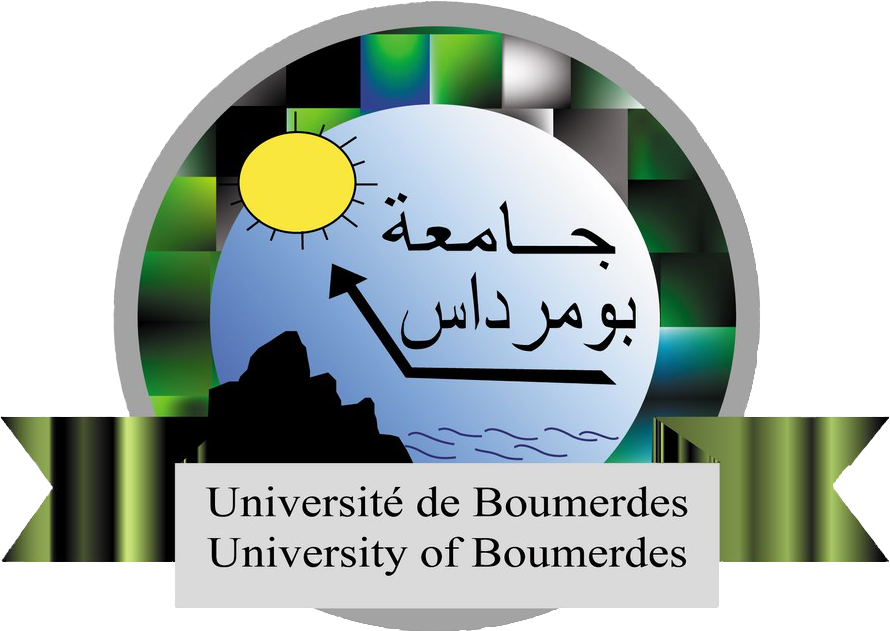
**University M’hamed Bougara Boumerdes**

**Institute of Electrical and Electronics Engineering (ex: INELEC)**



**Algorithms and Data structures laboratory**

LAB N°1

2/26/2022

**Superviser:**  **Students:**

Dr. A. Zitouni Madaoui Zakaria G3 Computer

Maallem Nassim G3 Computer

**Objectives:**

* Implementing the Hanoi tower problem
* Comparing the execution time of the recursive and iterative implementation when computing Fibonacci number of some natural number n.

**Task 1:** The Hanoi problem implementation

#include <iostream>

using namespace std;

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod);

int main()

{

int n = 4; *// Number of disks*

towerOfHanoi(n, 'A', 'B', 'C');

return 0;

}

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 0) return; *// Base condition*

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

cout << "Move disk " << n << " from rod " << from\_rod <<

" to rod " << to\_rod << endl;

towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

Running the above program for n=4 disks gave the following output.

❯ g++ hanoi.cpp -o hanoi

❯ ./hanoi

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

Move disk 4 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 2 from rod C to rod A

Move disk 1 from rod B to rod A

Move disk 3 from rod C to rod B

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

We can clearly see that the algorithm works correctly as all disk from rod A were transferred to rod B through rod C, according to the rules set at the beginning of the problem.

**Task 2:** Fibonacci number, recursive VS iterative

#include <iostream>

#include <ctime>

using namespace std;

long feb\_loop(int n);

long feb\_recursive(int n);

clock\_t start , end ;

int main(){

long n = 15;

long result;

clock\_t start;

clock\_t end;

cout << "Type a number: \n";

cin >> n;

start = clock(); *//measure recursive implementation start time*

result = feb\_recursive(n);

end = clock(); *//measure recursive implementation end time*

cout << "feb\_recursive of " << n << " is: " << result << "time taken is: "

<< ((float)(end - start) / CLOCKS\_PER\_SEC) << " seconds \n";

start = clock(); *//measure iterative implementation start time*

result = feb\_loop(n);

end = clock(); *//measure iterative implementation end time*

cout << "feb\_loop of " << n << " is: " << result << "| time taken is: "

<< ((float)(end - start) / CLOCKS\_PER\_SEC) << " seconds \n";

return 0;

}

long feb\_loop(int n){

if(n <= 1) return n;

long fib = 1;

long prevFib = 1;

for(int i=2; i<n; i++) {

long temp = fib;

fib+= prevFib;

prevFib = temp;

}

return fib;

}

long feb\_recursive(int n){

if(n == 0 || n== 1) return n;

return feb\_recursive(n - 1) + feb\_recursive(n-2);

}

Running this program which compares the the execution time of the recursive and iterative implementation of computing Fibonacci number for n = 45 produced the following output:

❯ ./feb

Type a number:

45

feb\_recursive of 45 is: 1134903170| time taken is: 11.8424 seconds

feb\_loop of 45 is: 1134903170| time taken is: 2e-06 seconds

**Conclusion:**

We can see a huge difference between the running time of the iterative and recursive algorithm. The recursive implementation was simpler and required less code, but it executed in 11.84 seconds. In contrast, the iterative implementation required few more lines of code but executed in almost no time ! (2e-6 seconds).

This experiment is also backed by the theory we covered in class. Analyzing the iterative implementation we can see that:

T1(n) = O(n-2) for n

And analyzing the recursive implementation we have

T2(n) = O(2n-1) for n

Where O(2n-1) grows exponentially and gets excessively large with large values of n, whereas the iterative implementation grows linearly O(n-2)